Transdichotomous Results in Computational Geometry, II: Offline Search*

Timothy M. Chan† Mihai Pătraşcu‡

October 12, 2010

Abstract

We reexamine fundamental problems from computational geometry in the word RAM model, where input coordinates are integers that fit in a machine word. We develop a new algorithm for offline point location, a two-dimensional analog of sorting where one needs to order points with respect to segments. This result implies, for example, that the convex hull of n points in three dimensions can be constructed in (randomized) time $n \cdot 2^{O(\sqrt{\lg\lg n})}$. Similar bounds hold for numerous other geometric problems, such as planar Voronoi diagrams, planar off-line nearest neighbor search, line segment intersection, and triangulation of non-simple polygons.

In FOCS'06, we developed a data structure for *online* point location, which implied a bound of $O(n\frac{\lg n}{\lg\lg n})$ for three-dimensional convex hulls and the other problems. Our current bounds are dramatically better, and a convincing improvement over the classic $O(n \lg n)$ algorithms. As in the field of integer sorting, the main challenge is to find ways to manipulate information, while avoiding the online problem (in that case, predecessor search).

1 Introduction

1.1 Sorting in Two Dimensions

Consider the following toy problem (in fact, a special case of offline point location), which we call *the slab problem*. We are given a vertical slab in the plane, m nonintersecting segments cutting across the slab, and n points in the slab. The goal is to identify the segment immediately below each of the n points. In other words, we would like to sort the points "relative to" the segments.

This is an appealing and natural generalization to two dimensions of the one-dimensional notion of sorting. It captures both an intuitive notion of ordering, and the non-orthogonal flavor so common in computational geometry. Indeed, as described below, an impressive collection of fundamental problems in computational geometry are known to be reducible to this simple problem, so there is a formal sense in which the slab problem is as central in computational geometry as sorting is in the one-dimensional world.

*A preliminary version of this work with the title "Voronoi Diagrams in $n \cdot 2^{O(\sqrt{\lg\lg n})}$ Time" appeared in *Proc. 39th ACM Symposium on Theory of Computing*, pages 31–39, 2007.

†School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca). This work has been supported by an NSERC grant.

‡AT&T Labs, Florham Park NJ, USA (mip@alum.mit.edu). Part of this work was done while the author was at MIT.

Classically, the slab problem is solved by binary searching among segments for each input point, for a cost of $O(\lg m)$ per point. This is optimal when one searches by binary decisions or assumes the input has infinite precision, as on a real RAM. However, a more reasonable assumption is that input has finite precision. We will assume, in particular, that all coordinates come from some universe $[2^w] = \{0, \ldots, 2^w - 1\}$, and that we are working on a word RAM with w-bit words (i.e. one coordinate fits in one word). See Section 1.3 for a discussion of these assumptions.

Until recently, successful use of the word RAM in computational geometry was limited to a restricted class of problems, especially problems involving orthogonal objects. However, in FOCS'06, we proposed improved data structures for the *online* slab problem, a problem of a fundamentally nonorthogonal nature [8]. The running time was asymptotically $\min\left\{\frac{\lg m}{\lg\lg m}, \sqrt{\frac{w}{\lg w}}\right\}$ per point. This represents a marginal improvement over $O(\lg m)$ for *any* universe, and a roughly quadratic improvement for small (polynomial) universes.

In the current paper, we describe an algorithm for the (offline) slab problem running in time $n \cdot 2^{O(\sqrt{\lg\lg m})} + O(m)$. Note that this bound does not depend on the universe (aside from assuming a coordinate fits in a word), and is deterministic. The bound is a dramatic improvement over our old bounds—note, for example, that the new bound grows more slowly than $n \lg^{\varepsilon} m + m$ for any constant $\varepsilon > 0$. In addition, the new bound represents a much more convincing improvement over the standard $O(n \lg m)$ bound, demonstrating the power granted by bounded precision.

The relation between our current algorithm and our results from [8] is best understood by a parallel to integer sorting. There, the online problem (predecessor search) is known to require comparatively large running times (e.g. in terms of n alone, an $\Omega(\sqrt{\frac{\lg n}{\lg\lg n}})$ lower bound per point is known [4]). Yet, one can find ways of manipulating information in the offline problem, such that the bottleneck of using the online problem is avoided (e.g. we can sort in $O(n\sqrt{\lg\lg n})$ expected time [14]). It should be understood that the purpose of this work is not to study "bit tricks" in the word RAM model, but to study how information about points and lines can be decomposed in algorithmically useful ways.

1.2 Applications

From [8] it follows that improved bounds for the slab problem lead to improved upper bounds for many fundamental problems in computational geometry [10, 11, 16, 17, 18]. We list some here. As before, the bounds do not depend on the universe for the coordinates. All the reductions below, except the last, are randomized.

1. We can compute the *convex hull* of n points in three dimensions in expected time $n \cdot 2^{O(\sqrt{\lg\lg n})}$. If the hull has H vertices, the bound can be reduced to $n \cdot 2^{O(\sqrt{\lg\lg H})}$.

2. We can compute the *Voronoi diagram* and the *Delaunay triangulation* of n points in the plane in expected time $n \cdot 2^{O(\sqrt{\lg\lg n})}$. As a consequence, we can also compute the *Euclidean minimum spanning tree* or solve the *largest empty circle* problem within the same time bound.

3. Given n red points and n blue points in the plane, we can compute the red point nearest to each blue point in expected time $n \cdot 2^{O(\sqrt{\lg\lg n})}$.

4. We can compute all K intersections of n line segments in the plane in expected time $n \cdot 2^{O(\sqrt{\lg\lg n})} + O(K)$. We can also construct the *trapezoidal decomposition* of the line segments within the same time bound.

5. We can triangulate a polygon with holes (or an environment with multiple disjoint polygons) with n vertices total in time $n \cdot 2^{O(\sqrt{\lg \lg n})}$.

Problems like convex hulls and Voronoi diagrams date back to the dawn of computational geometry, and for these problems standard $O(n \lg n)$ bounds have long been regarded as "optimal." The previous paper [8] has demonstrated that on the word RAM, $O(n \lg n)$ can be beaten slightly, by $O(n \frac{\lg n}{\lg \lg n})$ bounds. The current paper shows that $O(n \lg n)$ can be improved significantly.

Planar point location. The slab problem is actually a special case of the *offline planar point location* problem. Here, the input consists of a connected polygonal subdivision defined by a set of m segments in the plane, where segments are only allowed to touch at endpoints. Given n query points (offline), the goal is to identify the polygon (face) which contains each point.

In fact, reductions discussed above are to the offline planar point location problem. However, the general case of point location turns out to be reducible to the special case of the slab problem. In [8], we considered three different ways to achieve this reduction. These three approaches hold both in the offline and online cases. They all generate a multiplicative penalty of at most $O(\lg \lg m)$ per point, which is absorbed by our time bound, but differ in the cost per segment:

(i) random sampling gives a *randomized* $(n + m) \cdot 2^{O(\sqrt{\lg \lg m})}$ running time.

(ii) persistence and exponential trees give a deterministic $n \cdot 2^{O(\sqrt{\lg \lg m})} + O(\text{sort}(m))$ bound, where $\text{sort}(m)$ denotes the cost of sorting m integers.

(iii) planar separators are the most complicated (and least practical) strategy but give the best deterministic running time of $n \cdot 2^{O(\sqrt{\lg \lg m})} + O(m)$.

Higher dimensions. We can also solve the analog of the slab problem in any constant dimension. Instead of a slab, we are given a vertical prism, and instead of segments, we are given hyperplanes cutting across the prism where no two hyperplanes intersect inside the prism. The running time of our solution is $n \cdot 2^{O(\sqrt{\lg \lg m})} \lg^{1+o(1)} w + O(m)$. Although the bound now has an extra $\lg^{1+o(1)} w$ factor, this factor is relatively small. This result has a few applications as well, for example, to offline exact nearest neighbor search in higher dimensions and curve-segment intersection in two dimensions (see [8]).

Recent work. Subsequent to our conference publication, Buchin and Mulzer [6] announced a better result for planar Voronoi diagrams. They show that constructing Voronoi diagrams is equivalent to sorting, in the Word RAM augmented with one non-standard operation. In the standard Word RAM, they obtain an unconditional running time of $O(n\sqrt{\lg \lg n})$ by adapting the best known sorting algorithm of [14].

This improves the running time of all problems mentioned in item 2. above. However, this algorithm exploits special properties about Voronoi diagrams and nearest neighbor graphs, and does not imply improved results for the other problems considered here, such as 3-d convex hulls and offline point location.

1.3 Computational Geometry on a Grid

It is superfluous to state that bounded precision is a fact of life. Input data are given with finite precision and computers represent it internally in a bounded number of bits. Low-dimensional computational geometry has typically seen the negative consequences of this state of affairs. Algorithms are designed in idealized models of real arithmetic, and practitioners struggle to keep the algorithms working with imperfect precision.

Yet, there is also a good side to bounded precision, and our result shows it can be used to achieve significantly better algorithms. Again, we emphasize that our improved bound is independent of whatever the bound on the precision happens to be. We just require that coordinates can be manipulated in constant time, which has always been a standard assumption.

The philosophical question that we wish to address briefly is whether these benefits of bounded precision should be explored in theory. We believe firmly they should, examining the question both with an eye to practice and to theory.

Practice. From a theoretical perspective, the classic solutions to online point location using linear space and logarithmic query time would seem attractive. However, as pointed out in a survey on the topic [20], the most efficient and popular practical implementations do not use them. Instead, they use grid-pruning heuristics, not unlike some of our ideas. Thus, we can hope to gain theoretical understanding for what is already known to be effective in the real world—a standard goal for theory.

Turning this around, we can hope that theoretical improvements will suggest new ideas with an impact in practice. As presented, our results are theoretical because of large hidden constants, in both the slab problem and subsequent reductions. However, since the improvement over $O(n \lg n)$ is now quite significant, we find it plausible that some of the techniques developed here can provide inspiration for useful practical "tricks." A key step would be to circumvent the reductions and apply our techniques directly to target problems.

Theory. Even at theory's end of computational geometry, the assumption of bounded precision has been used fairly often. Unfortunately, this body of work (see the bibliographies of [8]) has been plagued by close ties to one-dimensional problems. For example, two-dimensional convex hulls can be found in linear time once points are sorted by x-coordinate. More typically, the problems considered involved orthogonal objects, and such orthogonal problems can more easily be decomposed into one-dimensional problems.

Our recent data structures for point location [8] broke this barrier, by presenting an improvement for a fundamentally nonorthogonal problem. The current paper tries to demonstrate that there are deeper questions to be explored in this direction of research. In our algorithm, we are forced to consider questions of decomposability and compressibility of information about two-dimensional objects, which seem fundamentally different from questions in one dimension. We feel this should have a theoretical appeal in itself.

For example, consider two standard tools in sorting. One is radix sort, which gives linear-time sorting in polynomial universes. Another is hashing, which is used in virtually all advanced RAM sorting algorithms, including [2, 3, 12, 13, 14, 15, 21]. In two dimensions, neither of these tools seems relevant. It is interesting that even without such basic tools, we can still obtain a rather efficient, determinstic algorithm (even outperforming some of the older RAM sorting results).

1.4 Overview

The remainder of this paper is organized as follows. In Section 2, we describe a simple algorithm for the slab problem, running in time $O(n\sqrt{\lg m}+m)$. This demonstrates the basic divide-and-conquer strategy behind our solution. In Section 3, we implement this strategy much more carefully to obtain an interesting recurrence that ultimately leads to the stated time bound of $n \cdot 2^{O(\sqrt{\lg \lg m})} + O(m)$. The challenges faced by this improvement are similar to issues in integer sorting, and indeed we borrow (and build upon) some tools from that field.

Unfortunately, the implementation of Section 3 requires a nonstandard word operation. In Section 4, we describe how to implement the algorithm on a standard word RAM, using only addition, multiplication, bitwise-logical operations, and shifts. Interestingly, the new implementation requires some new geometric observations that affect the design of the recursion itself.

2 An Initial Algorithm for the Slab Problem

2.1 The Basic Recursive Strategy

We begin with a recursive strategy based on a simple observation, taken from [8]. (Later in Section 4, we will replace this with a more complicated recursive structure.) In the following, the notation $\prec$ refers to the belowness relation.

Observation 1. *Fix b and h. Let S be a set of m sorted disjoint segments, where all left endpoints lie on an interval I_L of length 2^{ℓ_L} on a vertical line, and all right endpoints lie on an interval I_R of length 2^{ℓ_R} on another vertical line. In $O(b)$ time, we can find $O(b)$ segments $s_0, s_1, \ldots \in S$ in sorted order, which include the lowest segment of S, such that:*

(1) *for each i, at least one of the following holds:*

- (1a) *there are at most m/b segments of S between s_i and s_{i+1}.*
- (1b) *the left endpoints of s_i and s_{i+1} lie on a subinterval of length $2^{\ell_L - h}$.*
- (1c) *the right endpoints of s_i and s_{i+1} lie on a subinterval of length $2^{\ell_R - h}$.*

(2) *there exist segments $\tilde{s}_0, \tilde{s}_2, \ldots$ cutting across the slab, satisfying all of the following:*

- (2a) $s_0 \prec \tilde{s}_0 \prec s_2 \prec \tilde{s}_2 \prec \cdots$.
- (2b) *distances between the left endpoints of the $\tilde{s}_i$'s are all multiples of $2^{\ell_L - h}$.*
- (2c) *distances between right endpoints are all multiples of $2^{\ell_R - h}$.*

Proof: Let B contain every $\lfloor m/b \rfloor$-th segment of S, starting with the lowest segments s_0. Impose a grid over I_L consisting of 2^h subintervals of length $2^{\ell_L - h}$, and a grid over I_R consisting of 2^h subintervals of length $2^{\ell_R - h}$. We define s_{i+1} inductively based on s_i, until the highest segment is reached. We let s_{i+1} be the highest segment of B such that either the left or the right endpoints of s_i and s_{i+1} are in the same grid subinterval. This will satisfy (1b) or (1c). If no such segment above s_i exists, we simply let s_{i+1} be the successor of s_i, satisfying (1a). (See Figure 1.)

Let $\tilde{s}_i$ be obtained from s_i by rounding each endpoint to the grid point immediately above (ensuring (2b) and (2c)). By construction of the s_i's, both the left and right endpoints of s_i and s_{i+2} are in different grid subintervals. Thus, $\tilde{s}_i \prec s_{i+2}$, ensuring (2a). □

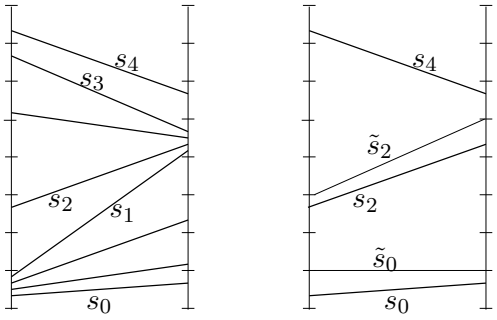

Figure 1: Proof of Observation 1: an example (with $\ell_L = \ell_R$). The left diagram shows the segments in B.

SLAB(Q, S):

0. if $m = 0$, set all answers to NULL and return
1. let $s_0, s_1, \ldots$ be the $O(b)$ segments from Observation 1
2. let φ be the projective transform mapping I_L to $\{0\} \times [0, 2^h]$ and I_R to $\{2^h\} \times [0, 2^h]$.
   Compute ROUND$(\varphi(Q))$ and $\varphi(\tilde{s}_0), \varphi(\tilde{s}_2), \ldots$
3. SLAB$_0$(ROUND$(\varphi(Q))$, $\{\varphi(\tilde{s}_0), \varphi(\tilde{s}_2), \ldots\}$)
4. for each $q \in Q$ with ANS[ROUND$(\varphi(q))$] $= \varphi(\tilde{s}_i)$ do
   set ANS$[q]$ = the segment from $\{s_{i-4}, \ldots, s_{i+7}\}$ immediately below q
5. for each s_i do
   SLAB$(\{q \in Q \mid \text{ANS}[q] = s_i\}, \{s \in S \mid s_i \prec s \prec s_{i+1}\})$

Figure 2: A recursive algorithm for the slab problem. Parameters b and h are fixed in the analysis; ROUND$(\cdot)$ maps a point to its nearest integral point.

The above observation naturally suggests a recursive algorithm. In the pseudocode in Figure 2, the input is a set Q of n points and a sorted set S of m disjoint segments, where the left and right endpoints lie on intervals I_L and I_R of length 2^{ℓ_L} and 2^{ℓ_R} respectively. At the end, $\text{ANS}[q]$ stores the segment from S immediately below q for each $q \in Q$. A special NULL value for $\text{ANS}[q]$ signifies that q is below all segments. We assume a (less efficient) procedure $\textsc{Slab}_0(Q, S)$, with the same semantics as $\textsc{Slab}(Q, S)$, which is used as a bottom case of the recursion. The choice of $\textsc{Slab}_0()$ is a crucial component of the analysis.

We first explain why the pseudocode works. In step 2, an explicit formula for the transform φ has already been given in [8]; this mapping preserves the belowness relation. According to property (2) in Observation 1, we know that the transformed segments $\varphi(\tilde{s}_0), \varphi(\tilde{s}_2), \ldots$ all have h-bit integer coordinates from $[2^h]$. After rounding, the n points $\varphi(Q)$ will lie in the same universe.

Any unit square can intersect at most two of the $\varphi(\tilde{s}_i)$'s, since these segments have vertical separation at least one and thus horizontal separation at least one (as slopes are between -1 and 1). If $\varphi(\tilde{s}_i) \prec \text{ROUND}(\varphi(q)) \prec \varphi(\tilde{s}_{i+2})$, then we must have $\varphi(\tilde{s}_{i-4}) \prec \varphi(q) \prec \varphi(\tilde{s}_{i+6})$, implying that $s_{i-4} \prec \tilde{s}_{i-4} \prec q \prec \tilde{s}_{i+6} \prec s_{i+8}$. Thus, at step 4, $\text{ANS}[q]$ contains the segment from $s_0, s_1 \ldots$ immediately below q. Once this is determined for every point $q \in Q$, we can recursively solve the subproblem for the subset of points and segments strictly between s_i and s_{i+1} for each i, as is done at step 5. An answer $\text{ANS}[q] = \text{NULL}$ from the i-th subproblem is interpreted as $\text{ANS}[q] = s_i$.

Let $\ell = (\ell_L + \ell_R)/2$ ($\ell \leq w$). Denote by $T(n, m, \ell)$ the running time of $\textsc{Slab}()$, and $T_0(n, b', h)$ the running time of the call to $\textsc{Slab}_0()$ in step 3. Steps 1, 2 and 4 can be implemented naively in $O(n + m)$ time. We have the recurrence:

$$T(n, m, \ell) \;=\; T_0(n, b', h) \;+\; O(n + m) \;+\; \sum_{i=0}^{b'} T(n_i, m_i, \ell_i), \tag{1}$$

where $b' = O(b)$, $\sum_i n_i = n$, $\sum_i m_i = m - b'$. Furthermore, according to property (1) in Observation 1, for each i we either have $m_i \leq \frac{m}{b}$ or $\ell_i \leq \ell - \frac{h}{2}$. This implies that the depth of the recursion is $O(\log_b m + \frac{\ell}{h})$.

2.2 An $O(n\sqrt{\lg m} + m)$ Algorithm

In the previous paper [8], we have noticed that for $b'h \approx w$, $\textsc{Slab}_0()$ can be implemented in $T_0(n, b', h) = O(n)$ time by packing b' segments from an h-bit universe into a word. By setting $b \approx \log^{\varepsilon} m$ and $h \approx w/\log^{\varepsilon} m$, this leads to an $O((n + m)\frac{\lg m}{\lg \lg m})$ algorithm.

Instead of packing multiple segments in a word, our new idea is to pack *multiple points* in a word. To understand why this helps, remember that the canonical implementation for $\textsc{Slab}_0()$ runs in time $O(n \lg m)$ by choosing the middle segment and recursing on points above and below this segment. By packing t segments in a word, we can hope to reduce this time to $O(n \log_t m)$. However, by packing t points in a word, we can potentially reduce this to $O(\frac{n}{t} \lg m)$, a much bigger gain. (One can also think of packing both points and segments, for a running time of $O(\frac{n}{t} \log_t m)$. Since we will ultimately obtain a much faster algorithm, we ignore this slight improvement.)

To implement this idea, step 2 will pack $\text{ROUND}(\varphi(Q))$ with $O(w/h)$ points per word. Each point is alotted $O(h)$ bits for the coordinates, plus $\lg b = O(h)$ bits for the answer $\text{ANS}[\text{ROUND}(\varphi(q))]$ which $\textsc{Slab}_0()$ must output. This packing can be done in $O(n)$ time, adding one point at a time.

Working on packed points, $\textsc{Slab}_0()$ has the potential of running faster, as evidenced by the following lemma. For now, we do not concern ourselves with the implementation on a word RAM, and assume nonstandard operations (an operation takes two words as input, and outputs one word).

Lemma 2. *If $\lg b \le h \le w$, $\textsc{Slab}_0()$ can be impemented on a RAM with nonstandard operations with a running time of $T_0(n, b, h) = O(n\frac{h}{w}\lg b + b)$.*

Proof: Given a segment and a number of points packed in a word, we can postulate two operations which output the points above (respectively below) the segment, packed consecutively in a word. Choosing a segment, we can partition the points into points above and below the segment in $O(\lceil n\frac{h}{w}\rceil)$ time. In the same asymptotic time, we can make both output sets be packed with $\lfloor\frac{w}{h}\rfloor$ points per word (merging consecutive words which are less than full).

We now implement the canonical algorithm: partition points according to the middle segment and recurse. As long as we are working with $\ge \frac{w}{h}$ points, the cost is $O(\frac{h}{w})$ per point for each segment, and each point is considered $O(\lg b)$ times. If we are dealing with less than $\frac{w}{h}$ points, the cost is $O(1)$, and that can be charged to the segment considered. Thus, the total time after packing is $O(n\frac{h}{w}\lg b + b)$.

The last important issue is the representation of the output. By the above, we obtain the sets of points which lie between two consecutive segments. We can then trivially fill in the answer for every point in the $\lg b$ bits alotted for that. However, we want an array of answers for the points in the original order. To do that, we trace the algorithm from above backwards in time. We use an operation which is the inverse of splitting a word into points above and below a segment. □

Plugging the lemma into (1), we get $T(n, m, \ell) = O(n\frac{h}{w}\lg b + n + m)\cdot O(\log_b m + \frac{\ell}{h})$. Setting $\lg b = \sqrt{\lg m}$ and $h = w/\sqrt{\lg m}$, we obtain $T(n, m, w) = O((n+m)\sqrt{\lg m})$. This can be improved to $O(m+n\sqrt{\lg m})$ by the standard trick of considering only one in $O(\sqrt{\lg m})$ consecutive segments. For every point, we finish off by binary searching among $O(\sqrt{\lg m})$ segments, for a negligible additional time of $O(n \lg\lg m)$.

3 An $n \cdot 2^{O(\sqrt{\lg\lg m})} + O(m)$ Algorithm

3.1 Preliminaries

To improve on the $O(m + n\sqrt{\lg m})$ bound, we *bootstrap*: we use an improved algorithm for $\textsc{Slab}()$ as $\textsc{Slab}_0()$, obtaining an even better bound for $\textsc{Slab}()$. To enable such improvements, we can no longer afford the $O(n)$ term in the recurrence (1). Rather, a call to $\textsc{Slab}()$ is passed Q in word-packed form, and we want to implement the steps between recursive calls in *sub*linear time (close to the number of words needed to represent Q, not to $n = |Q|$).

This task will require further ideas and more sophisticated word-packing tricks. To understand the complication, let us contrast implementing steps 2 and 5 of $\textsc{Slab}()$ in sublinear time. Computing $\textsc{round}(\varphi(Q))$ in Step 2 is solved by applying a function in parallel to a word-packed vector of points. This is certainly possible, at least using nonstandard word operations. However, step 5 needs to group elements of Q into subsets (i.e. sort Q according to $\textsc{ans}[q]$). This is a deeper information-theoretic limitation, and it is rather unlikely that it can always be done in time linear in the number of words needed to store Q. The problem has connections to applying permutations in external memory, a well-studied problem which is believed to obey similar limitations [1].

To implement step 5 (and also step 4), we will use a subroutine $\textsc{Split}(Q, \textsc{label})$. This receives a set Q of ℓ-bit elements, packed in $O(n\frac{\ell}{w})$ words. Each elements $q \in Q$ has a $(\lg m)$-bit label $\textsc{label}[q]$ with $\lg m \le \ell$. The labels are stored in the same $O(n\frac{\ell}{w})$ words. We can think of each word as consisting of two portions, the first containing $O(\frac{w}{\ell})$ elements and the second containing the corresponding $O(\frac{w}{\ell})$ labels. The output of $\textsc{Split}(Q, \textsc{label})$ is a collection of sublists, so that all elements of Q with the same label are put in the same sublist (in arbitrary order).

In addition, we will need $\textsc{Split}()$ to be reversible. Suppose the labels in the sublists have been modified. We need a subroutine $\textsc{Unsplit}(Q)$, which outputs Q in the original order before $\textsc{Split}()$, but with the modified labels attached.

The following lemma states the time bound we will use for these two operations. The implementation of $\textsc{Split}()$ is taken from a paper by Han [12] and has been also used as a subroutine in several integer sorting algorithms [13, 14]. As far as we know, the observation that $\textsc{Unsplit}()$ is possible in the same time bound has not been stated explicitly before.

Lemma 3. *Assume* $\textsc{label}[q] \in [m]$ *for all* $q \in Q$*, and let* M *be a parameter. If* $\frac{w}{\ell} \lg m \le \frac{1}{2} \lg M$ *and* $\lg m \le \ell \le w$*, both* $\textsc{Split}()$ *and* $\textsc{Unsplit}()$ *require time* $O(n\frac{\ell}{w} \lg \frac{w}{\ell} + M)$.

Proof: Let $g = \frac{w}{\ell}$. Each word contains g elements, with $g \lg m$ bits of labels. Put words with the same label pattern in the same bucket. This can be done in $O(n/g + \sqrt{M})$ time, since the number of different label patterns is at most $2^{g \lg m} \le \sqrt{M}$. For each bucket, we form groups of g words and *transpose* each group to get g new words, where the i-th element of the j-th new word is the j-th element of the i-th old word. Transposition can be implemented in $O(\lg g)$ standard word operations [21]. Elements in each new word now have identical labels. We can put these words in the correct sublists, in $O(n/g + m)$ time. There are at most g leftover elements per bucket, for a total of $O(\sqrt{M}g) = o(M)$; we can put them in the correct sublists in $o(M)$ time. The total time is therefore $O((n/g) \lg g + M)$.

To support unsplitting, we remember information about the splitting process. Namely, whenever we transpose g words, we create a record pointing to the g old words and the g new words. To unsplit, we examine each record created and transpose its g new words again to get back the g old words (with labels now modified). We can also update the leftover elements by creating $o(M)$ additional pointers. □

A particularly easy application of this machinery is to implement the algorithm of Section 2 with standard operations (with a minor $\lg \lg m$ slowdown). This result is not interesting by itself, but it will be used later as the base case of our bootstrapping strategy.

Corollary 4. *If* $\frac{w}{h} \lg b \le \frac{1}{2} \lg M$ *and* $\lg b \le h \le w$*, the algorithm for* $\textsc{Slab}_0()$ *from Lemma 2 can be implemented on a word RAM with standard operations in time* $T_0(n, b, h) = O(n\frac{h}{w} \lg b \lg \frac{w}{h} + bM)$.

Proof: The nonstandard operations used before were splitting and unsplitting a set of points packed in a word, depending on sidedness with respect to a segment. It is not hard to compute sidedness of all points from a word in parallel using standard operations: we apply the linear map defining the support of the segment to all points (which is a parallel multiplication and addition), and keep the sign bits of each result. The sign bits define 1-bit labels for the points, and we can apply $\textsc{Split}()$ and $\textsc{Unsplit}()$ for these. □

Since the algorithm is used with $b = \sqrt{\lg m}$ and $h = w/\sqrt{\lg m}$, we incur a slowdown of $O(\lg \frac{w}{h}) = O(\lg \lg m)$ per point compared to the implementation with nonstandard operations. By setting

$M = m^2$, the algorithm of the previous section would then run in time $O(n\sqrt{\lg m}\lg\lg m + m^3)$ if implemented with standard operations. (The dependence of the second term on m can be lowered as well.)

3.2 The Improved Algorithm

Our fastest algorithm follows the same pseudocode of Figure 2, but with a more careful implementation of the individual steps. Let $\widetilde{\ell}$ be the number of bits per point and $\widetilde{m}$ the original number of segments in the root call to SLAB(). We have $\lg \widetilde{m} \le \widetilde{\ell} \le w$. In a recursive call to SLAB(), the input consists of some n points and $m \le \widetilde{m}$ segments, all with coordinates from $[2^\ell]$, where $\ell \le \widetilde{\ell}$. Points will be packed in $O(\widetilde{\ell})$ bits each, so the entire set Q occupies $O(n\frac{\widetilde{\ell}}{w})$ words. At the end, the output ANS$[q]$ is encoded as a label with $\lg \widetilde{m}$ bits, stored within each point $q \in Q$, with the order of the points unchanged in the list Q. Note that one could think of repacking more points per word as ℓ and m decrease, but this will not yield an asymptotic advantage, so we avoid the complication (on the other hand, repacking before the call to SLAB$_0$() is essential).

In step 2, we can compute ROUND$(\varphi(Q))$ in time linear in the number of words $O(n\frac{\widetilde{\ell}}{w})$, by using a nonstandard word operation that applies the projective transform (and rounding) to multiple points packed in a word. Unfortunately, it does not appear possible to implement this efficiently using standard operations. We will deal with this issue in Section 4, by changing the algorithm for SLAB() so that we only require affine transformations, not projective transformations.

Before the call to SLAB$_0$() in step 3, we need to condense the packing of the points ROUND$(\varphi(Q))$ to take up $O(n\frac{h}{w})$ words. Previously, we had $O(\frac{w}{\ell})$ points per word, but after step 2, only $O(h)$ bits of each point were nonzero. We will stipulate that points always occupy an number of bits which is a power of 2. This does not affect the asymptotic running time. Given this property, we obtain a word of ROUND$(\varphi(Q))$ by condensing $\widetilde{\ell}/h$ words. This operation requires shifting each old word, and ORing it into the new word.

Note that the order of ROUND$(\varphi(Q))$ is different from the order of Q, but this is irrelevant, because we can also reverse the condensing easily. We simply mask the bits corresponding to old word, and shift them back. Thus, we obtain the labels generated by SLAB$_0$() in the original order of Q. Both condensing and its inverse take $O(n\frac{\widetilde{\ell}}{w})$ time.

For the remainder of the steps, we need to SPLIT() and UNSPLIT(). For that, we fix a parameter M satisfying $\frac{w}{\ell}\lg\widetilde{m} \le \frac{1}{2}\lg M$. In step 4, we first split the list ROUND$(\varphi(Q))$ into sublists with the same ANS labels. For each sublist, we can perform the constant number of comparisons per point required in step 4, and then record the new ANS labels in the list, in time linear in the number of words $O(n\frac{\widetilde{\ell}}{w})$. It is standard to implement this in the word RAM by parallel multiplications (see the proof of Lemma 3). To complete step 4, we UNSPLIT() to get back the entire list ROUND$(\varphi(Q))$, and then copy the ANS labels to the original list Q in $O(n\frac{\widetilde{\ell}}{w})$ time. Since both lists are in the same order, this can be done by masking labels and ORing them in.

To perform step 5, we again split Q into sublists with the same ANS labels. After the recursive calls, we unsplit to get Q back in the original order, with the new ANS labels.

3.3 Analysis

For $\frac{w}{\ell}\lg\widetilde{m} \le \frac{1}{2}\lg M$ and $\lg\widetilde{m} \le \widetilde{\ell} \le w$, the recurrence (1) now becomes:

$$T(n,m,\ell) \;=\; T_0(n,b',h) \;+\; O\left(n\frac{\widetilde{\ell}}{w}\lg\frac{w}{\widetilde{\ell}} + M\right) \;+\; \sum_{i=0}^{b'} T(n_i,m_i,\ell_i), \tag{2}$$

where $b' = O(b)$, $\sum_i n_i = n$, $\sum_i m_i = m - b'$, and for each i, we either have $m_i \le \frac{m}{b}$ or $\ell_i \le \ell - \frac{h}{2}$. As before, the depth of the recursion is bounded by $O(\log_b \widetilde{m} + \frac{\widetilde{\ell}}{h})$.

Assume that for $\frac{w}{h}\lg b \le \frac{1}{2}\lg M$ and $\lg b \le h \le w$, an algorithm with running time

$$T_0(n,b,h) \;\le\; c_k\left(n\frac{h}{w}\lg^{1/k} b\;\lg\left(\frac{w}{h}\lg b\right) \;+\; bM\right)$$

is available to begin with. This is true for $k = 1$ with $c_1 = O(1)$ by Corollary 4.

Then the recurrence (2) yields:

$$T(n,\widetilde{m},\widetilde{\ell}) \;=\; O(c_k)\cdot\left(\left[n\frac{h}{w}\lg^{1/k} b\;\lg\left(\frac{w}{h}\lg b\right) \;+\; n\frac{\widetilde{\ell}}{w}\lg\frac{w}{\widetilde{\ell}}\right]\cdot\left(\log_b\widetilde{m} + \frac{\widetilde{\ell}}{h}\right) \;+\; mM\right).$$

Set $\lg b = \lg^{k/(k+1)}\widetilde{m}$ and $h = \widetilde{\ell}/\lg^{1/(k+1)}\widetilde{m}$. Notice that indeed $\frac{w}{h}\lg b = \frac{w}{\widetilde{\ell}}\lg\widetilde{m} \le \frac{1}{2}\lg M$ and $\lg b \le h \le w$. Thus, we obtain an algorithm with running time:

$$T(n,\widetilde{m},\widetilde{\ell}) \;\le\; c_{k+1}\left(n\frac{\widetilde{\ell}}{w}\log^{1/(k+1)}\widetilde{m}\;\lg\left(\frac{w}{\widetilde{\ell}}\lg\widetilde{m}\right) \;+\; \widetilde{m}M\right)$$

for some $c_{k+1} = O(1)\cdot c_k$.

Iterating this process k times, we get:

$$T(n,\widetilde{m},\widetilde{\ell}) \;\le\; 2^{O(k)}\left(n\frac{\widetilde{\ell}}{w}\lg^{1/k}\widetilde{m}\;\lg\left(\frac{w}{\widetilde{\ell}}\lg\widetilde{m}\right) \;+\; \widetilde{m}M\right)$$

for any value of k. Choosing $k = \sqrt{\lg\lg\widetilde{m}}$ to asymptotically minimize the expression, and plugging in $\widetilde{\ell} = w$ and $M = \widetilde{m}^2$ (so that indeed $\frac{w}{\widetilde{\ell}}\lg\widetilde{m} \le \frac{1}{2}\lg M$), we get:

$$T(n,\widetilde{m},w) \;=\; 2^{O(\sqrt{\lg\lg\widetilde{m}})}\,(n \;+\; \widetilde{m}^3).$$

We can reduce the dependence on $\widetilde{m}$ to linear as follows. First, select one out of every $\widetilde{m}^{3/4}$ consecutive segments of S, and run the above algorithm on just these $\widetilde{m}^{1/4}$ segments. This takes time $2^{O(\sqrt{\lg\lg\widetilde{m}})}(n + \widetilde{m}^{3/4})$ time. Now recurse between each consecutive pair of selected segments. The depth of the recursion is $O(\lg\lg\widetilde{m})$, and it is straightforward to verify that the running time is $n\cdot 2^{O(\sqrt{\lg\lg\widetilde{m}})} + O(\widetilde{m})$.

4 Avoiding Nonstandard Operations

The only nonstandard operation used by the algorithm of Section 3 is applying a projective transform in parallel to points packed in a word. Unfortunately, it does not seem possible to implement

this in constant time using standard word RAM operations (since, according to the formula for projective transform, this operation requires multiple divisions where the divisors are all different).

One idea is to simulate the special operation in slightly superconstant time. We can use the circuit simulation results of Brodnik et al. [5] to reduce the operation to $\lg w \cdot (\lg \lg w)^{O(1)}$ standard operations. For the version of the slab problem in dimension 3 or higher, this is the best approach we know. Note that the results from the previous section hold in any constant dimension, by simply using the multidimensional analog of Observation 1 from [8].

However, in two dimensions we can get rid of the dependence on the universe, obtaining a time bound of $n \cdot 2^{O(\sqrt{\lg \lg m})} + O(m)$ on the standard word RAM. This constitutes the object of this section.

4.1 The Center Slab

By horizontal translation, we can assume the left boundary of our vertical slab is the y-axis. Let the abscissa of the right boundary be Δ. For some h to be determined, let the *center slab* be the region of the plane defined by $\Delta/2^h \le x \le \Delta \cdot (1 - 2^{-h})$. The lateral slabs are defined in the intuitive way: the left slab by $0 \le x \le \Delta/2^h$ and the right slab by $\Delta \cdot (1 - 2^{-h}) \le x \le \Delta$.

The key observation is that distances are somewhat well-behaved in the center slab, so we will be able to decrease both the left and right intervals at the same time, not just one of them. This enables us to use (easier to implement) affine maps instead of projective maps. Center slabs were also used in one of our previous papers [19], but as presented there, the idea cannot get rid of the dependence on the universe. This paper's definition and use of the center slab is rather different, and gets rid of this dependence.

The following is a replacement for Observation 1:

Observation 5. *Fix b and h. Let S be a set of m sorted disjoint segments, such that all left endpoints lie on an interval I_L and all right endpoints lie on an interval I_R, where both I_L and I_R have length 2^ℓ. In $O(b)$ time, we can find $O(b)$ segments $s_0, s_1, \ldots \in S$ in sorted order, which include the lowest segment of S, such that:*

(1) *for each i, at least one of the following holds:*

  (1a) *there are at most m/b segments of S between s_i and s_{i+1}.*

  (1b) *both the left and right endpoints of s_i and s_{i+1} are at distance at most $2^{\ell-h}$.*

(2) *there exist segments $\tilde{s}_0 \prec \tilde{s}_1 \prec \cdots$ cutting across the slab, satisfying all of the following:*

  (2a) *distances between the left endpoints of the $\tilde{s}_i$'s are multiples of $2^{\ell-2h}$.*

  (2b) *ditto for the right endpoints.*

  (2c) *inside the center slab, $s_0 \prec \tilde{s}_0 \prec s_2 \prec \tilde{s}_2 \prec \cdots$.*

Proof: Let B contain every $\lfloor m/b \rfloor$-th segment of S, starting with the lowest segment s_0. We define s_{i+1} inductively. If the next segment after s_i has either the left or right endpoints at distance greater than $2^{\ell-h}$, let s_{i+1} be this segment, which satisfies (1a). Otherwise, let s_{i+1} be the *highest* segment of B which satisfies (1b).

Now impose grids over I_L and I_R, both consisting of 2^{2h} subintervals of length $2^{\ell-2h}$. We obtain $\tilde{s}_i$ from s_i by rounding each endpoint to the grid point immediately above. This immediately

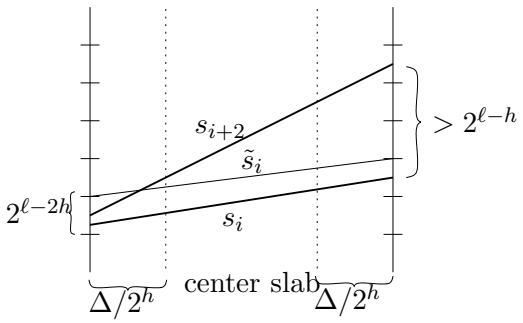

Figure 3: Proof of Observation 5: a center slab.

implies $\tilde{s}_0 \prec \tilde{s}_1 \prec \cdots$ and $s_i \prec \tilde{s}_i$. Unfortunately, $\tilde{s}_i$ and s_{i+k} may intersect for arbitrarily large k (e.g. $s_i, \ldots, s_{i+k}$ are very close on the left, while each consecutive pair is far on the right). However, we will show that inside the center slab, $\tilde{s}_i \prec s_{i+2}$. (See Figure 3.)

By construction, s_i and s_{i+2} are vertically separated by more than $2^{\ell-h}$ either on the left or on the right. Since lateral slabs have a fraction of 2^{-h} of the width of the entire slab, the vertical separation exceeds $2^{\ell-h}/2^h = 2^{\ell-2h}$ anywhere in the center slab. Rounding s_i to $\tilde{s}_i$ represents a vertical shift of less than $2^{\ell-2h}$ anywhere in the slab. Hence, $\tilde{s}_i \prec s_{i+2}$ in the center slab. □

We now describe how to implement Slab(), assuming the intervals containing the left endpoints (I_L) and the right endpoints (I_R) both have length 2^ℓ. In this section, we only deal with points in the center slab. It is easy to Split() Q into subsets corresponding to the center and lateral slabs, and Unsplit() at the end.

We use Observation 5 instead of Observation 1. Since I_L and I_R have equal length, the map φ is affine. Thus, it can be implemented using parallel multiplication and parallel addition. This means step 2 can be implemented in time $O(n\frac{\tilde{\ell}}{w})$ using standard operations.

Because we only deal with points in the center slab, and there $s_i \prec \tilde{s}_i \prec s_{i+2}$ (just like in the old Observation 1), steps 4 and 5 work in the same way.

4.2 Lateral Slabs

To deal with the left and right slabs, we use the following simple observation, which we only state for the left slab by symmetry. Note that the guarantees of this observation (for the left slab) are virtually identical to that of Observation 5 (for the center slab). Thus, we can simply apply the algorithm of the previous section for the left and right slabs.

Observation 6. *Fix b and h. Let S be a set of m sorted disjoint segments, such that all left endpoints lie on an interval I_L and all right endpoints lie on an interval I_R, where both I_L and I_R have length 2^ℓ. In $O(b)$ time, we can find $O(b)$ segments $t_0, t_1, \ldots \in S$ in sorted order, which include the lowest segment of S, such that:*

(1) *for each i, at least one of the following holds:*

(1a) *there are at most m/b segments of S between s_i and s_{i+1}.*

(1b) *anywhere in the left slab, the vertical separation between s_i and s_{i+1} is less than $2^{\ell-h+1}$.*

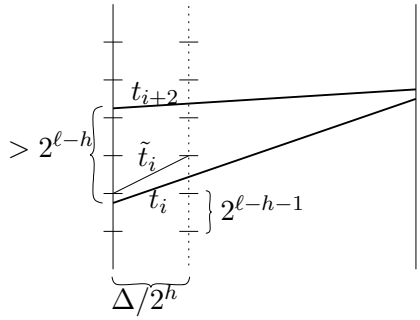

Figure 4: Proof of Observation 6: a left slab.

(2) *there exist segments* $\tilde{t}_0 \prec \tilde{t}_1 \prec \cdots$ *cutting across the left slab, satisfying all of the following:*

(2a) *distances between the left endpoints of the* $\tilde{t}_i$*'s are multiples of* $2^{\ell-2h}$*.*

(2b) *ditto for the right endpoints.*

(2c) *inside the left slab,* $t_0 \prec \tilde{t}_0 \prec t_2 \prec \tilde{t}_2 \prec \cdots$*.*

Proof: Let I_A be the vertical interval at the intersection of the right edge of the left slab with the parallelogram defined by I_L and I_R. Note I_A also has size 2^ℓ.

Let B contain every $\lfloor m/b \rfloor$-th segment of S, starting with the lowest segment t_0. Given t_i, we define t_{i+1} to be the highest segment of B which has the left endpoint at distance at most $2^{\ell-h}$ away. If no such segment above t_i exists, let t_{i+1} be the successor of t_i in B (this will satisfy (1a)). In the first case, (1b) is satisfied because the right endpoints of t_i and t_{i+1} are at distance most 2^ℓ, so on I_A, the separation is at most $2^{\ell-h}(1-2^{-h}) + 2^\ell \cdot 2^{-h} < 2^{\ell-h+1}$.

Now impose grids over I_L and I_A, both consisting of 2^{h+1} subintervals of length $2^{\ell-h-1}$. We obtain $\tilde{t}_i$ from t_i by rounding the points on I_L and I_A to the grid point immediately above. Note that the vertical distance between t_i and $\tilde{t}_i$ is less than $2^{\ell-h-1}$ anywhere in the left slab. On the other hand, the left endpoints of t_i and t_{i+2} are at distance more than $2^{\ell-h}$. The distance on I_A (and anywhere in the left slab) is at least $2^{\ell-h}(1-2^{-h}) \geq 2^{\ell-h-1}$. Thus $t_i \prec \tilde{t}_i \prec t_{i+2}$. (See Figure 4.) □

4.3 Bounding the Dependence on m

Our analysis needs to be modified, because segments are simultaneously in the left, center and right slabs, so they are included in 3 recursive calls. In other words, in recurrence (2), we have to replace $\sum_i m = m - b'$ with a weaker inequality $\sum_i m \leq 3m$. Recall that for our choice of b and h, the depth of the recursion is bounded by $O(\log_b \widetilde{m} + \frac{\widetilde{\ell}}{h}) = O(\lg^{1/(k+1)} \widetilde{m})$. Thus, the cost per segment is increased by an extra factor of $3^{O(\lg^{(1/(k+1)} \widetilde{m})} = 3^{O(\sqrt{\lg \widetilde{m}})}$ for each bootstrapping round; the cost per point does not change. With $k = \sqrt{\lg \lg \widetilde{m}}$ rounds, the overall dependence on $\widetilde{m}$ is now increased slightly to $2^{O(\sqrt{\lg \lg \widetilde{m}})} \cdot \widetilde{m}^3 \cdot 3^{O(\sqrt{\lg \widetilde{m} \lg \lg \widetilde{m}})} = O(\widetilde{m}^{3+\varepsilon})$. As before, this can be made $O(\widetilde{m})$ by working with $\widetilde{m}^{1/4}$ segments and recursing.

5 Open Problems

Though our algorithm for offline point location is deterministic, the reductions to other problems [8] introduce randomization. It would be nice to derandomize them. Another interesting question is whether some of these reductions also hold in the other direction. For example, the trapezoidal decomposition problem of disjoint line segments is clearly no easier than offline point location, but can the same be said for other problems like 3-d convex hulls?

A problem with an $O(n \lg n)$ upper bound that we currently cannot improve is *counting* intersections between a set of red segments and a set of blue segments, given that no segments of the same color intersect [9]. Note that this problem is no easier than counting inversions in a permutation, which takes $O(n\sqrt{\lg n})$ time by the best known algorithm [7].

Finally, the complexity of the online point location problem remains open: can the $O(\frac{\lg n}{\lg \lg n})$ and $O(\sqrt{\frac{w}{\lg w}})$ upper bounds from [8] be improved, or can stronger lower bounds be proved?

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[2] A. Andersson. Faster deterministic sorting and searching in linear space. In *Proc. 37th IEEE Sympos. Found. Comput. Sci.*, pages 135–141, 1996.

[3] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *J. Comput. Sys. Sci.*, 57:74–93, 1998.

[4] P. Beame and F. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Sys. Sci.*, 65:38–72, 2002.

[5] A. Brodnik, P. B. Miltersen, and J. I. Munro. Trans-dichotomous algorithms without multiplication—some upper and lower bounds. In *Proc. 5th Int. Workshop Algorithms Data Struct.*, pages 426–439, London, UK, 1997. Springer-Verlag.

[6] K. Buchin and W. Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. In *Proc. 50th IEEE Sympos. Found. Comput. Sci.*, pages 139–148, 2009.

[7] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. 21st ACM/SIAM Sympos. Discrete Algorithms*, pages 161–173, 2010.

[8] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.*, 32(10):703–729, 2010. Preliminary versions appeared in FOCS'06.

[9] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[11] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[12] Y. Han. Improved fast integer sorting in linear space. *Inf. Comput.*, 170:81–94, 2001.

[13] Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. In *Proc. 34th ACM Sympos. Theory Comput.*, pages 602–608, 2002.

[14] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 135–144, 2002.

[15] D. G. Kirkpatrick and S. Reisch. Upper bounds for sorting integers on random access machines. *Theoret. Comput. Sci.*, 28:263–276, 1984.

[16] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[17] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[19] M. Pătraşcu. Planar point location in sublogarithmic time. In *Proc. 47th IEEE Sympos. Found. Comput. Sci.*, pages 325–332, 2006.

[20] J. Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 767–785. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[21] M. Thorup. Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42:205–230, 2002.